NLP - Project 1

CRF - Named Entity Recognition

Abstract

The project involved implementation of Conditional Random Fields (CRF) sequence tagger for the task of Named Entity recognition. Viterbi decoding is implemented on an HMM model and we generalize the same for the CRF forward - backward algorithm and perform learning and inference on the same. The data used in this project is derived from the CoNLL 2003 Shared Task on Named Entity Recognition (Tjong Kim Sang and De Meulder, 2003).

1 Introduction

Named-entity recognition (NER) involves classifying named entity mentions in a given piece of text into pre-defined categories such as the person names, organizations, locations etc. We used Hidden Markov Models (HMM) which can be used to predict a sequence of states given a sequence of input tokens. HMMs are used in the field of NLP to predict the sequence of BIO tags given an input sentence. But HMMs assume the inputs are independent of each other and to overcome this drawback, we go for CRFs which study the dependence between input features.

1.1 Hidden Markov Models

HMMs can be used to train models and tag each token in a sequence with a label, where the model comprises of initial, emission and transition probabilities. Dynamic programming is used where we recursively compute the most likely subsequence of states (i.e tags) that account for the first t observations and this is referred to as the Viterbi algorithm. We also record back pointers in each step to backtrack to the most probable state sequence. The Viterbi algorithm is implemented with scores that come from log probabilities to find the highest log-probability path.

1.2 Conditional Random Fields

CRF is a discriminative model for tagging tasks that uses arbitrary features of the input. They have a globally normalized form and take into account the dependence among the features. In CRFs we omit the input tokens x and consider feature functions f. Marginal probabilities are computed with dynamic programming using forward - backward. We accumulate gradient over all the emissions and transitions, where we try to maximize the negative log likelihood and perform corresponding weight updates.

2 Implementation Details

Our task is to predict BIO tags for every word in a given sequence, where BIO represents begin, inside and outside tag. For example, consider the name Barack Obama, in which Barack is tagged as B-PER and Obama is tagged as I-PER. Any word that is not a named entity is tagged as O (i.e outside). An I tag always succeeds the corresponding B tag in any given sequence. We have 9 unique tags in the given data set.

The final state is the most probable state sequence and we follow the back pointers till the initial state for constructing the full sequence.

2.1 Part 1 : Viterbi Decoding

Viterbi algorithm is used to predict the most likely sequence of hidden states (tag sequence) given a model and set of observations. The model comprises of initial sequence log probabilities, transition log probabilities which gives the transition probability of transitioning from state s_i to s_j and emission log probabilities that gives the probability of observing o_j from state s_i. At every step, the back pointers are also kept track of for each cell to know how that cell's value was derived and we backtrack to get the predicted tag sequence.

Computing the viterbi scores:

$$v_1(j) = a_0 b_j(o_1), 1 <= j <= N$$

$$v_t(j) = \max_{i=1}^{n} a_{ij} b_j(o_t), 1 <= j <= N, 1 < t <= T$$

$$P^* = v_{T+1}(S_F) = \max_{i=1}^{n} v_T(i) a_{iF}$$

Computing the viterbi back pointers:

$$bt_1(j) = s_0, 1 <= j <= N \quad (1)$$

$$bt_t(j) =_{i=1}^{n} v_{t-1}(i) a_{ij} b_j(o_t), 1 <= j <= N, 1 < t <= T \quad (2)$$

$$q_T^* = bt_{T+1}(S_F) =_{i=1}^{n} v_T(i) a_{iF} \quad (3)$$

2.2 Part 2 : CRF

For training the model, we were provided with a set of input word features such as the word, POS tag, n-gram and other context features which include the features of the current, previous and next tag. Using a sparse feature set, results in a very large vector space as the size of vocabulary is too large. Hence we index the features and map them to axes. We implement the forward - backward algorithm where we compute intermediate values of alpha and beta used for the computation of the marginal probabilities.

Forward Algorithm:

$$\alpha_1(s) = exp(\phi_e(s, 1, x))$$

$$\alpha_t(s_t) = \sum_{s_t - 1} \alpha_{t-1}(s_t - 1) exp(\phi_e(s_t, t, x))$$

Backward Algorithm:

$$\beta n(s) = 1 \beta_t(s_t) = \sum_{s_t + 1} \beta_{t+1}(s_t + 1) exp(\phi_e(s_t + 1, t + 1, x))$$

Computing the marginal probability:

$$P(y_s|x) = \frac{forward_i(s) backward_i(s)}{\sum_{s'} forward_i(s') backward_i(s')}$$

In the above equation, the transition probabilities have not been specified, however we can include them too. All the computation has been performed in log space to avoid underflow conditions. After the marginal probabilities have been computed, we apply gradient ascent to update our feature weights accordingly and the unregularized adagrad optimizer has been used for this. Different set of optimizers have been provided such as SGD, unregularized Adagrad, and L1-regularized Adagrad which use the access and score methods to update weights by the apply gradient update, which take as input a counter of feature values for this gradient. The transition probabilities from the HMM model are used and during inference we enforce hard constraints stating transitions that are illegal (e.g. B-PER $\rightarrow$ I-ORG is illegal). The Counter class provided has also been used for maintaining sparse maps.

2.3 Part 3 : Extension

As part of the extension, additional features were included to train the model to see if the accuracy could be improved. The accuracies have been summarized in table 1

2.3.1 Word Counter

One of the features include a word counter, where the count of every word in the training set is fed as an input to the model. Since most of the NER tags are unique names, their frequency would be lower when compared to other POS tags such as verbs and prepositions.

2.4 Stop Word Removal

Stop words were removed both when indexing entries for searching and when retrieving them as the result of a search query. They include common verbs, prepositions etc. and they don't help us to understand the true meaning of a sentence. This can save memory space and processing time. We use the stop words list provided in the Natural Language ToolKit (NLTK).

2.5 Lemmatization

Lemmatization refers to returning the base/ root of any given word using a vocabulary and by performing morphological analysis of words, normally aiming to remove inflectional endings only and to return the dictionary form of a word, which is known as the lemma. The Wordnet Lemmatizer provided in the NLTK has been used for lemmatization which returns meaningful base words and uses the Wordnet Database to do the same. Using lemmas reduces the feature set size as most of the words are mapped to a single base word.

2.5.1 TF-IDF

In addition, the TF-IDF feature was also used which corresponds to the term-frequency, inverse document frequency of every word in the training set. The TF-IDF vectorizer provided in the scikit learn library was used for the same. Since most of the tags in the data set are unique names, their inverse document frequency would be high and hence their tf-idf score is also comparatively high when compared with other common words in the data set.

3 Training and Evaluation

The model is trained with various sizes of the training data set and the accuracy is found to grad-

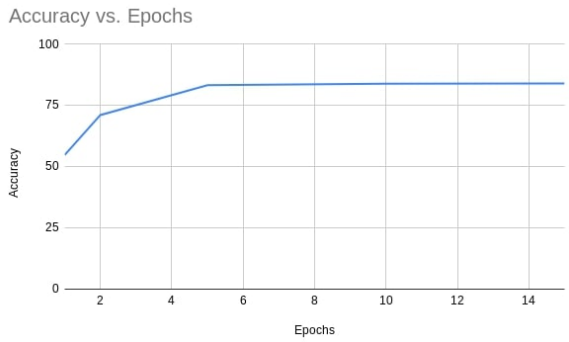

Figure 1: Epochs vs Accuracy

Figure 2: Training data size vs Accuracy

ually improve with increase in the data set size and also with increase in the number of epochs. As we see in the above Figure 1, it can be observed that the model learns quite well from the first few epochs and the accuracy almost remains the same after that. As we increase the feature size, the accuracy of the model too increases. Including the word count as a separate feature doesn't seem to improve the accuracy much. If we map the word count to separate indices, the model wouldn't learn much but having a threshold value and words occurring more than or less than the threshold frequency can be specified in the feature vector. Once the model was trained, the weights are saved separately and these are used for testing the model. Adding the term frequency and inverse document frequency feature increased the training time though stop word removal was performed on top of it. Most of the additional features that were included would be more useful for tasking such as summarization, question-answering where we try to understand the underlying context, but in the problem we are currently solving since the model has to learn only named entities, these features don't result in a significant increase in the accuracy.

Features Used	Accuracy
Base Features	83.87
Base Features + Lemmatization	83.21
Base Features + Stop Word Removal	84.20
Base Features + TF-IDF	80.03
All features	81.22

Table 1: Accuracy on Dev Dataset

Since most of the names are tagged as NNP or NNPS, we get 70% accuracy by considering only the part of speech tag of the word. Since most of the names are followed by verbs and hence we consider the next word and their corresponding POS tag too. Another experiment on HMM and CRF was performed where we train the model by varying the size of training data. As shown in Figure 2, we can infer that increasing the size of the training data, drastically improves the accuracy and due to the various advantages of CRFs over HMM, the accuracy of CRFs is better than Markov models.

4 Results

As we see in the Figure 2, it can be observed that the model learns from few epochs and the accuracy almost remains the same after that as we increase the feature size, the accuracy of the model too increases. An F1 score of 84 was obtained using the CRF model and F1 of 76.89 for performing inference using Viterbi decoding.

5 Conclusion

An accuracy of 76.89% is achieved on the development set on HMM model using Viterbi decoding with a precision and recall of 85.4 and 79.9 respectively. For the implementation of CRFs, an accuracy of 84% is achieved on the development set with the feature set provided. The model was also trained by giving additional features and we see that certain features don't improve the accuracy of the model whereas some do. Since CRFs take into account the dependence between words and we feed into the model various word features, which is not possible in HMMs, we see that CRFs perform much better than HMMs.

Collaborator - Madhumitha Sakthi